Statistical Inference for Networks

First Practical: R, and network statistics

1. What is R?

R is an open source language and environment for statistical computing and graphics. It provides a wide variety of statistical and graphical techniques, and is easy to extent. R compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

Installing the program under Windows

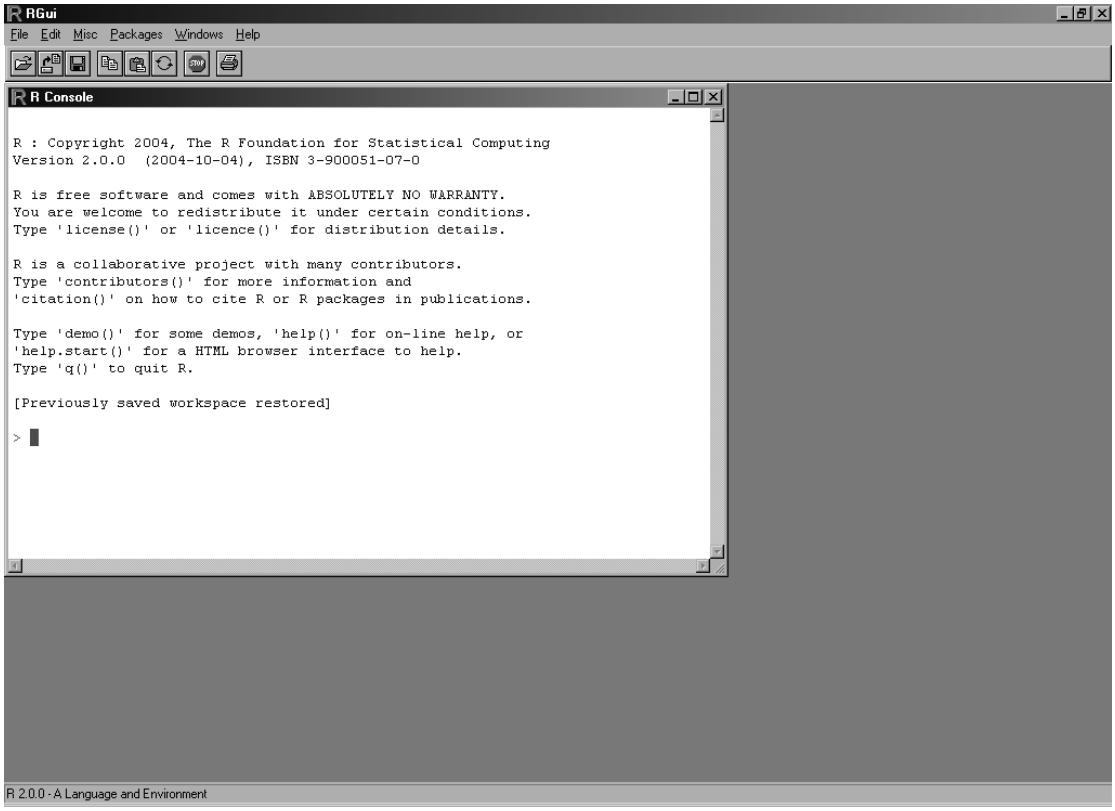
To set things up, create a new directory on your hard disk into which you can download your data files. To get the free package, go to

www.stats.bris.ac.uk/R/

and download by following the sequence

Windows → base → Download R 2.8.1 for Windows

Once you have done this you can run R-2.8.1-win32.exe which will install the program and place an icon on your desktop. Now start up the program and you should see something like the following screen.



You can quit the program at any time by clicking on File Exit at the top left of the screen.

R commands

It is important to realise that R is case sensitive so that, for example, A and a would be regarded as different symbols. Care is therefore needed when typing in commands.

Individual commands may be separated either by a semi-colon (;) or by a new line (i.e.by hitting <return>). Comments can be put in anywhere by starting with a hashmark (#); everything to the end of the line will then be a comment. If a command is not complete by the end of the line (i.e. when you hit <return>), the prompt at the next line will be
+
and will continue on further, subsequent lines until the command is syntactically complete. This is very handy because it means that you can enter a very long command without having to run over the screen width by taking several lines. It also means that, if you fail to enter, say, a closing bracket, it will simply keep prompting you until you do enter it.

Command lines can be recalled and edited by using the up and down arrow keys to scroll through them. Once a command is located in this way, the horizontal arrow keys can be used to edit it (<del> is used to delete and the other keys are used to add in text) and the <return> key to execute it.
An expression is evaluated, printed and then discarded.

Vectors, assignment, and matrices

R works on data structures which are identified by having names, the simplest such structure being a data vector (i.e. an ordered collection of numbers). Suppose you want to set up a vector x comprising the numbers 3.2, 5.1, 1.4, 2.3, 6.8, 19.7. Simply type in

```
x <- c(3.2, 5.1, 1.4, 2.3, 6.8, 19.7)
```

This assigns the vector to x using the function c(). Typing in the assignment operator, which is the two characters <- (i.e. "less than" followed by "hyphen") causes the vector to be received by x, and the operator can be viewed as an arrow which sends it there. The arrow can point either way, so it does not matter which way round you make this assignment. Entering

```
c(3.2, 5.1, 1.4, 2.3, 6.8, 19.7) -> x
```

has exactly the same result. Try doing this and then enter

```
x
```

The result will be that x will be printed on screen. If you were to enter

```
1/x
```

the reciprocals of the six values will be printed to screen. Note that these are shown on-screen and then lost. If you want to assign the reciprocals to, say, y, you enter

```
y <- 1/x
```

The c() function is used for concatenating, so entering

```
z <- c(x, 0, -x)
```

produces a vector with 13 entries consisting of the six values, a zero in the middle, and the six values made negative. Vectors can also be used in doing arithmetic. If you enter

```
 2 + 3
[1] 5
```

you can see that the numbers are added. If you enter vectors

```
 x + y
[1] 3.512500 5.296078 2.114286 2.734783 6.947059 19.750761
```

you get the vectors added element by element.

```
z <- 2*x + y + 1
```

will over-write the z you created earlier with a new vector with the i_{th} element being $2x_i + y_i + 1$.

To create a matrix, we also use the c command, but we have to specify the number of rows and (or) columns. If you enter

```
m <- matrix( c(1,2,2,1), nr=2)
```

You have created a 2 by 2 matrix. To get at the individual elements, use

```
m[1,2]
```

to obtain the (1,2) element of m. You can manipulate matrices fairly easily, for example

```
m^2
```

gives the square of the matrix m.

You may want to save your work by copying it into a script file, which is a drop-down option from the menu.

Getting help with R

R is easy to begin to use but somewhat more difficult to master. If at least you remember the name of the function you need to use, type help(functionname), as in

```
help(help)
```

It is also good to know that most documentation includes a "see also" section, so if you can think of a function that is similar to the one you want, sometimes "see also" can be helpful. If you don't know the name of the function, here are two alternatives:

```
help.search("network") # Search for anything on the topic of "networks"
help.start() # Start the interactive help browser
```

Finally, there are many R introductions on the web, even at the R home page under "documentation". Just try googling "R introduction".

Installing packages

Packages from R can be installed using Windows pull-downs menus or the R functions install.packages and installDataPackage. The relevant package for today is *igraph.*

See also the introduction by David Hunter at

http://www.stat.psu.edu/~dhunter/Rnetworks/

and by Sandrine Dutoit and Nicholas J Newell at

http://www.stat.berkeley.edu/~sandrine/Docs/Talks/MBI04/Lects/lectLab1BioCRIntro.pdf

for further information.

2. Network Statistics

2.1. Getting the data ready

For this practical we shall use as data set a set of Yeast protein interaction, available at

http://www.stats.ox.ac.uk/~reinert/dtc/networks.html,

First we load the package which we shall need: From the drop-down menu packages, select “load package”, then select *igraph*.

Now we look at the Yeast data. The data are in the so-called .net format, which comes from another network package called Pajek. The .net format is supported by *igraph*, so type

```
yeast<-read.graph("http://www.stats.ox.ac.uk/~reinert/dtc/YeastL.net", format="pajek")
```

and our network data are ready to use. Trying to plot the data using

```
plot.igraph(yeast)
```

is not very successful; try instead

```
plot.igraph(yeast, layout=layout.lgl)
```

for a plot where we can see something.

2.2 Calculating network summaries

The *igraph* package provides commands which give our network summaries fairly instantly. For the degree distribution:

```
degree.distribution(yeast)
```

gives the degree distribution for the yeast data: a vector where the first element is the relative frequency of zero degree nodes, the second with degree one, and so on. You can check that the relative frequencies add to 1;

```
sum(degree.distribution(yeast))
[1] 1
```

The command

```
degree(yeast)
```

gives the individual degrees;

```
mean(degree(yeast))
```

gives the average degree. To get a histogram of the degree distribution, we can use

```
hist(degree.distribution(yeast))
```

The clustering coefficient is called transitivity in *igraph*; the command is

```
transitivity(yeast)
```

The command

```
average.path.length(yeast)
```

gives the average shortest path length for the Yeast data.

There are a number of other network summaries which you could try out, have a look at the *igraph* package documentation at

http://stat.ethz.ch/CRAN/web/packages/igraph/index.html

3. Generating random networks

The *igraph* package makes it easy to simulate from different random graph models; the commands often and with .game.

3.1 Erdös-Renyi graphs and mixtures

To generate an Erdös-Renyi random graph with 10,000 nodes and edge probability 1/1000, or 2/1000, we use

```
er1<- erdos.renyi.game(10000, 1/1000)
er2<-  erdos.renyi.game(10000, 2/1000)
```

Let's look at their clustering coefficients:

```
transitivity(er1)
transitivity(er2)
```

Hold on – should they not be equal to the edge probability?

We can also compare their degree distributions. In R the function

```
par(mfrow=c(2,1))
```

generates an array of two plots in the same display. Try

```
par(mfrow=c(2,1))
hist(degree.distribution(er1))
hist(degree.distribution(er2))
```

You may also be interested in

```
mean(degree(er1))
mean(degree(er2))
```

To generate an Erdös-Renyi mixture with two types, we have to specify the number of nodes, the number of types, the type distribution, and the matrix of probabiliies (which should be symmetric). Here are two examples (nr stands for the number of types)

```
pf <- matrix( c(1, 0, 0, 1), nr=2)
pr1 <- preference.game(20, 2, pref.matrix=pf)

pf2 <- matrix( c(1, 2, 2, 1), nr=2)
pr2 <- preference.game(20, 2, type.dist=rep(1, 2),  pref.matrix=pf2)
```

You can also try

```
pr3<- preference.game(20, 2, type.dist=c(5,15),  pref.matrix=pf2)
```

Feel free to compare their network summaries!

3.2. Watts-Strogatz small worlds

For a Watts-Strogatz network we have to specify the dimension of the lattice, the number of nodes, the size of the neighbourhood, nad the rewiring probability; we can use the command

```
ws1<-watts.strogatz.game(1, 100, 5, 0.5)
```

where 1 is the dimension, 100 is the number of nodes, 5 is the neighbourhood size, and 0.5 is the rewiring probability.

3.3. Barabasi-Albert networks

For a Barabasi-Albert network on 10,000 nodes with preferential attachment proportional to the node degree, we use

```
bar1<-barabasi.game(10000)
```

For a Barabasi-Albert network on 10,000 nodes with preferential attachment proportional to the square of the node degree, we use

```
bar2<-barabasi.game(10000, power=2)
```

3.4 Model comparison

Now we can use the network statistics to compare the different models. Here are some questions to consider.

1. What is the effect of choosing different parameters in the models?
2. How do the node degree distributions differ in the different models?
3. How does the clustering coefficient vary across the models?

This part is open-ended.

Assignment

Pick your favourite summary statistic and examine how its distribution changes, following the steps below.

1. Across three different models which have roughly the same expected number of edges;
2. In your favourite network model, looking at three different choices for your model parameter(s).
3. Make sure that you simulate enough graphs (100 graphs for each part of the question may suffice) so that you trust that your results are not purely due to chance.

Compile your results in a short write-up, describing what you did. One page would suffice. The assignment will be marked according to the levels fail – pass – distinction. Sufficient for a pass would be a report which would indicate the changes by examples and/or plots.

The assignment is due Monday April 27 at 9:30 am. You can either email the report to reinert@stats.ox.ac.uk, or hand it to Lisa Bligh.